Benchmarks and Software Standards: A Case Study of GARCH Procedures

DRAFT: not to be quoted; all results subject to change.

B. D. McCullough[1]
Federal Communications Commission
2000 M St. NW Room 533
Washington, DC 20554
INTERNET: bmccullo@fcc.gov
and
C. G. Renfro
Alphametrics Corporation

[1]Thanks for comments to C. Cummins, G. Fiorentini, C. Hallahan, M. Kim, and R. Schoenberg. The views expressed herein are those of the authors and do not necessarily reflect those of their employers.

This version is GARCH3.TEX on 1 April 1999.

1 Introduction

It is well-known that two different packages can produce two different sets of solutions to the same estimation problem. Sometimes the discrepancies can be attributed to implementation, as in the treatment of corrections for AR(1) errors (Lovell and Selover, 1994) which can take several forms, including Cochrane-Orcutt, Prais-Winsten and Beach-MacKinnon. Other times no such possible reason for the discrepancies can be identified, which constitutes *prima facie* evidence that at least one of the packages is inaccurate. In principle, a benchmark can resolve the question of accuracy: Silk's (1996) use of Calzalori and Panattoni's (1988) FIML benchmark is an example of this. However, benchmarks can be of more use than determining the accuracy of software; they can also assist in setting standard features which econometric software should possess, such as defaults and options for ific econometric procedures. This often-overlooked function of benchmarks has become more important in recent years with the proliferation of sophisticated nonlinear econometric procedures.

Thirty years ago econometric software incorporated techniques which were essentially linear in nature. In such a world, it was possible to focus benchmark evaluation on the issue of algorithmic implementation, as demonstrated by the Longley (1967) benchmark. But as econometric software began to incorporate nonlinear estimation, the creation of benchmarks, never an easy task, became technically more difficult. Nevertheless, there could still be widespread agreement as to how a benchmark model should be specified and how it should be estimated, *e.g.*, Donaldson and Schnabel (1987). In the present day, nonlinear procedures are so complex (GMM, ARIMA, GARCH, etc.) that the question of how a benchmark should be specified no longer is clear-cut. For such procedures a benchmark cannot be merely a set of exceedingly accurate estimates. A broader context is required.

For complex econometric procedures, benchmark models should be chosen with two questions in mind, one positive and the other normative: What models *can* software packages estimate? and What models *should* they be able to estimate? This positive/normative distinction is the crux of the larger context in which benchmark models need to be considered. At first sight it might seem that the literature can provide appropriate guidance. Ostensibly, "What models can be estimated?" should be evident from the package's documentation. What models should be able to be estimated is surely just a matter of consulting the econometrics literature. However, in the first case, user manuals tend to be less than comprehensive in their treatment of computational details, especially for the more sophisticated econometric procedures. In the second case, determining the set of estimatable models actually depends upon the degree of consensus: if an authoritative survey has been written or the procedure is so well known as to be treated as a matter of course in textbooks, there is likely to be a standard specification; arguably, all packages should be able to estimate such a model – but whether they can is a different question. And even in the case of standard procedures, there are likely to be both subtle and not-so-subtle variations that may have statistical or numerical consequences.

Considering the creation of a standardized benchmark, there are several issues that must be resolved. Frequently, it will be necessary to carefully determine, and possibly restrict, the range of candidate estimation options. A person writing a benchmark program might even select a single specification and then estimate that model with benchmark precision. But of course, making this choice imposes the requirement that at least some of the packages tested must be able to estimate this model – otherwise, how might any package's results be compared to the benchmark. This requirement would seem to restrict the choice to only standard specifications, if any such exist. However, for there to be a standard specification essentially implies a well developed literature, which also raises the question, How do we know packages are reliable during the process of the infancy or adolescence of a procedure? In fact, benchmarks that point to the significant questions are needed at each stage of a procedure's progressive adoption. In this way, benchmarks can be of more use than simply measuring accuracy: they can also help set standards for software implementations of econometric procedures, which is, incidentally, an important part of replicability (Dewald, Thursby, and Anderson, 1986; Feigenbaum and Levy, 1993; Renfro, 1997, pp. 279-80).

With these considerations in mind, this paper attempts to benchmark the GARCH procedures in several software packages. Since our focus is software packages and benchmarks in general, we do not identify specific packages by name. To fix ideas, Section Two describes the types of problems encountered when using these packages to conduct default estimation of a simple GARCH model. Section Three then discusses the basics of GARCH estimation, with attention to computational details and sources of numerical error. Section Four present a benchmark for GARCH estimation due to Fiorentini, Calzolari and Panattoni (1996, hereafter FCP). Section Five applies the benchmark to the several packages. Section Six presents the conclusions.

2 Default Estimation

An obvious empirical requirement for a benchmark is an appropriate dataset. Such a dataset should be well-known and readily available. Bollerslev and

Ghysel's (1996, hereafter BG) 1974 observations on the daily percentage nominal returns for the Deutschemark/British pound exchange rate can be used (the "BG data," which are available at the *Journal of Business and Economic Statistics* archive). All packages should be able to estimate the simplest of all GARCH models, a constant with GARCH(1,1) errors. In many papers the model is presented as

$$\begin{aligned} y_t &= \mu + \epsilon_t \\ \text{where} & \\ \epsilon_t|\Phi_{t-1} \sim N(0, h_t) & \\ \text{and} & \\ h_t &= \alpha_0 + \alpha_1 \hat{\epsilon}_{t-1}^2 + \beta_1 \hat{h}_{t-1} \end{aligned}$$

We note two important features of this model. First, it maximizes not the likelihood, but the *conditional* likelihood. Second, this is only a partially-specified model, because the elements on which the likelihood is conditioned are not specified: initial values for $\hat{\epsilon}_0$ and $\hat{h}_0$ are not given. In fact, the initialization of the series $\hat{\epsilon}_t$ and $\hat{h}_t$, though often overlooked, can substantially affect the "solution" produced by the software. Two common methods of initialization are: (1) discarding observations; and (2) estimating the unconditional expectation of the series. All packages implement a default initialization, and so can estimate a GARCH(1,1) model, but not all packages mention what initialization is used – this is a serious omission.

For the above simple model, Table 1 presents default coefficient estimates for the seven packages we have evaluated. There appears to be some agreement, but the need for a benchmark is apparent – the difference cannot be attributed to rounding error. One source of discrepancy is that the packages are not all maximizing the same GARCH function. Different conditional

package	μ	α_0	α_1	β_1
X1	-0.00540	0.0096	0.142	0.821
X2	-0.00608	0.0098	0.144	0.818
X3	-0.00624	0.0108	0.153	0.806
X4	-0.00619	0.0108	0.152	0.806
X5	-0.00613	0.0107	0.153	0.806
X6	-0.00919	0.0098	0.144	0.818
X7	-0.00619	0.0108	0.153	0.806

Table 1: GARCH estimates

likelihood functions, which arise from different initializations for $\hat{\epsilon}_0^2$ and $\hat{h}_0$, lead naturally to different parameter estimates. Another possible source of discrepancy is numerical error, which is discussed in detail in the appendix.

At this point, is it useful to pose a very general question. Initially, one might think that there is a simple GARCH likelihood well-established in the literature and that all packages could estimate it by invoking the appropriate options. In such a case, it would be possible to attribute discrepancies solely to numerical error. There is such a GARCH likelihood, but most packages cannot estimate it because they do not have options to allow its estimation. Thus the question is raised, What options should GARCH procedures have?

Consider also Table 2, which presents default t-statistics on the coefficients. Here there is less agreement, and the need for a benchmark is even more apparent. In addition to the above-mentioned sources of discrepancy, there are two more. First, the differing parameter estimates can lead to different standard error estimates. Second, there are at least five different methods for computing standard errors for GARCH coefficients, which can be computed with varying degrees of accuracy.

In order to consider the various methods, let $g(\theta)$ and $H(\theta)$ be the gradient and Hessian of the likelihood function, respectively, and let Q be an

package	μ	α_0	α_1	β_1
X1	-0.64	8.01	11.09	53.83
X2	-0.72	3.80	5.80	26.34
X3	-0.74	8.13	10.95	48.64
X4	-0.74	8.15	10.97	48.61
X5	-0.73	5.58	7.91	36.96
X6	-1.08	8.09	10.77	45.90
X7	-0.67	1.66	2.86	11.12

Table 2: GARCH t-stats

estimator of the covariance matrix. The choice $Q = g(\theta)g(\theta)'$ produces the outer product of the gradient (OP) estimator which is the basis of the BHHH method. The BHHH method is the usual Gauss-Newton method applied to maximizing a likelihood rather than minimizing a sum of squares. Direct use of the Hessian (H) is $Q = -H(\theta)$ from the Newton-Raphson method. Choosing $Q = E[H(\theta)]$ produces the negative of the information matrix (IM) from the method of scoring. These are the three usual estimators for nonlinear procedures. In the context of maximum likelihood estimation, there are two others. The quasi-maximum likelihood (QMLE) estimator is $Q = H^{-1}(gg')H^{-1}$ and the Bollerslev-Wooldridge (BW) estimator is $Q = I^{-1}(g'g)I^{-1}$. The accuracy of these various covariance matrix estimators is affected not only by the accuracy of the coefficient estimates, but also by the method of calculating derivatives: using a finite-difference approximation to the Hessian, rather than an exactly calculated analytic Hessian, induces numerical error in the estimated standard errors of the coefficients.

It is worth noting that estimation schemes vary in the amount of derivative information used. For example, Newton-Raphson requires explicit calculation of the function value, $L(\theta)$, the gradient, and the Hessian. The BHHH method, by contrast, requires only explicit calculation of the function and the gradient, as do quasi-Newton methods such as Davidon-Fletcher-Powell (DFP) and Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithms. Moreover, the method by which derivatives are calculated matters. In particular, analytic derivatives are more accurate than numerical derivatives and lead to more accurate estimation (Bard, 1974, p. 117; Gill, Murray and Wright, 1981, p. 285).[1] The most accurate way to calculate $H(\theta)$, for example, is by analytic differentiation of an analytic gradient. This is more accurate than a finite-difference approximation based on an analytic gradient which, in turn, is more accurate than a finite-difference approximation based on a numerical gradient.

Some packages offer misleading information on this point, suggesting that the difference between numerical and analytic derivatives is only speed or a negligible loss of accuracy. Anonymous quotes from two user manuals demonstrate this. "[Numerical derivatives] provide sufficent accuracy to obtain appropriate solutions to the problems. However, it is relatively slow compared to analytic derivatives." "Analytic [derivatives] will, in general, be slightly more accurate than numeric derivatives, if the calculation of analytic derivatives has been carefully optimized by hand to remove common subexpressions." Both these statements are true only in specific cases, and should not be taken as general rules.

The general rule is that if the gradient is analytic, then the Hessian may be calculated numerically from the gradient with no loss of accuracy. This is supported by Monte Carlo evidence (Donaldson and Schnabel, 1987). However, this is only a general rule and it does not apply to GARCH estimation,

[1]It is true that, with proper programming, numerical derivatives can be practically as good as analytic derivatives. However, this proper programming entails more than the simple forward difference or even central difference approximations commonly found in statistical and econometric software. See Quandt (1983, pp. 731-735).

for which accurate estimation requires not only an analytic gradient, but an analytic Hessian, as well.

An analytic gradient leads to more accurate estimates than a numerical gradient (Bard, 1974, p. 117). Finite-difference approximation of the gradient should only be used when analytic derivatives are not available, either because there is no closed-form expression for the derivatives or the function is too complex to differentiate. Tables 1 and 3 obviously demonstrate the need for careful evaluation, especially since documentation sometimes is unclear and even incorrect. We will give specific examples of this.

3 The Basic GARCH Model

The GARCH model has many extensions, but the basic GARCH(p, q) model is given by

$$\begin{aligned} y_t &= x_t'b + \epsilon_t \qquad \epsilon_t|\Psi_{t-1} \sim N(0, h_t) &(1)\\ h_t &= \alpha_0 + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j h_{t-j} &(2) \end{aligned}$$

The conditional likelihood function for this model can be written as

$$\begin{aligned} L_t(\theta) &= \sum_{t=1}^{T} l_t(\theta) &(3)\\ l_t(\theta) &= -\frac{1}{2}\ln h_t - \frac{1}{2}\frac{\epsilon_t^2}{h_t} &(4) \end{aligned}$$

As given, the model is only partially specified and therefore not estimatable. In order to fully specify the model, the initialization of the series h_t and ϵ_t^2 must be defined. This is a particularly important implementation issue because, as can be seen from (4), these initial values enter the likelihood function. To be explicit, *ceteris paribus*, different initializations lead to different parameter estimates because different likelihoods are being maximized. We note that this is a separate issue from the determination of starting values for an iterative, nonlinear estimation procedure. Lumsdaine (1996, lemma 6) shows that dependence on the initialization is asymptotically neglible. Perhaps for this reason, many authors do not even discuss the initialization in their GARCH papers, though this is tantamount to not identifying the conditional likelihood function on which reported estimation results are based. It is intuitively obvious that reported results do depend on the initialization, since in finite samples the initialization is not necessarily negligible and can even be substantial, as shown by Diebold and Schuermann (1998). This is an important consideration when benchmarking accuracy is the objective.

Many choices for this initialization have been proposed in the literature. One such initialization is

$$h_t = \epsilon_t^2 = \frac{1}{T}\sum_{s=1}^{T}\epsilon_s^2, \qquad t \leq 0 \tag{5}$$

An initial vector of residuals for ϵ_s can be computed from a preliminary OLS regression.[2] It is not unreasonable to suggest that *every* GARCH procedure should be able to estimate a GARCH(1,1) model with the initialization (5). This model should always be supported for two reasons. First, because a minimal degree of flexibility is thereby insured. Second, the existence of the FCP GARCH benchmark allows users to verify the accuracy of their packages. The primary reason a package might not be able to estimate this model is because the developer has decided not to allow the user to control the initialization. Such packages are of questionable value, since the user cannot specify the GARCH likelihood to be maximized. Some packages have options for various initializations, but this is still unsatisfactory. The researcher

[2]It is unclear whether Bollerslev (1986) held SS fixed at the value provided by the preliminary regression, or whether SS was updated with current parameter estimates on each iteration. This demonstrates the need for a benchmark model at every stage of a procedure's development.

might well need an initialization for which the developer has not provided an option – thus, the need for flexibility is apparent. Other relevant package requirements are considered in the Appendix, which examines in greater detail the numerical aspects of nonlinear estimation, including circumstances that may cause estimation attempts to fail disastrously.

4 The Benchmark

It is well-known that the GARCH likelihood is a complicated and highly nonlinear function (Engle and Bollerslev, 1986, p. 24; Bollerslev, Engle, and Nelson, 1994, p. 2981), an implicit argument for the use of numerical derivatives. However, at least for the standard GARCH(1,1) model, analytic first derivatives are easily derived (Greene, 1993, §18.5). In a recent paper, Fiorentini, Calzolari and Panattoni (1996, FCP) paved the way for substantially more accurate GARCH estimation by providing readily useable, closed-form expressions for the first- and second-derivatives of the GARCH conditional likelihood. For their Monte Carlo study, they wrote FORTRAN code to estimate three types of GARCH models with analytic derivatives. Their Monte Carlo study reached three conclusions which are relevant for present purposes:

- For methods which use an analytic gradient and a Hessian, numerical calculation of the Hessian is inferior to analytic calculation of the Hessian. Moreover, numerical second derivatives tend to be unstable, and thus not suitable for calculating standard errors.

- The OP estimator of the standard error is inferior to methods based on second derivative information, especially the QML and BW estimates.

- Faster convergence is achieved by using one algorithm to start and another to finish than by using one algorithm throughout the iterative process. Such use of two different algorithms often is referred to as a "hybrid algorithm" or a "mixed method."

The FCP code, applied to the BG data, constitutes a benchmark. FCP's Model I, which corresponds roughly to Bollserslev's (1986) seminal formulation[3], is estimated:

$$y_t = \mu + \epsilon_t \text{ where } \epsilon_t|\Phi_{t-1} \sim N(0, h_t) \text{ and } h_t = \alpha_0 + \alpha_1 \hat{\epsilon}_{t-1}^2 + \beta_1 \hat{h}_{t-1}.$$

This needs six starting values for: $\mu, \alpha_0, \alpha_1, \beta_1$, $\hat{\epsilon}_0^2$ and $\hat{h}_0$. Two natural choices are: $\mu = -0.016427$, the unconditional mean of y; and $\alpha_0 = 0.221130$, such that $\alpha_0/(1-\alpha_1-\beta_1)$ equals the unconditional and variance of y. The values $\alpha_1 = 0.35$ and $\beta_1 = 0.50$ were chosen after examining several possibilities, all of which led to the same solution. The least squares regression of y_t on a constant yields a sum of squared errors which, divided by the sample size, produces the starting value for both $\hat{\epsilon}_0^2$ and $\hat{h}_0$, in accordance with Eq. 5. The FCP code uses $||\hat{\theta}^k - \hat{\theta}^{k-1}|| < \epsilon||\hat{\theta}^{k-1}||$ as a stopping rule with convergence tolerance $\epsilon = 10^{-9}$. Successful convergence is tested by examining whether the squared norm of the gradient is less than ϵ, *i.e.*, the convergence criterion is $||g(\theta)||^2 < \epsilon$. Results to six significant digits are presented in Table 3. We note that Bollerslev and Ghysels estimated the same model on the same data, and obtained the following results (QMLE standard errors in parentheses): $\mu = -0.006(0.009)$, $\omega = 0.264(0.075)$, $\alpha_1 = 0.153(0.054)$, and $\beta_1 = 0.806(0.073)$ where, in terms of the original model, $\omega = \alpha_0/(1-\alpha_1-\beta_1)$. These results are consistent with the FCP benchmark.

[3]Again, we note that Bollserslev might have held ϵ_0^2 and h_0 fixed, but this is unclear.

coefficient		standard error				
		-H	OP	QMLE	IM	BW
μ	-.619041E-2	.846212E-2	.843359E-2	.918935E-2	.837628E-2	.873092E-2
α_0	.107613E-1	.285271E-2	.132298E-2	.649319E-2	.192881E-2	.312364E-2
α_1	.153134E-0	.265228E-1	.139737E-1	.535317E-1	.194012E-1	.273219E-1
β_1	.805974E-0	.335527E-1	.165604E-1	.724614E-1	.218399E-1	.301509E-1

Table 3: FCP GARCH Benchmark

5 Applying the Benchmark

Relying on defaults does not permit the packages to be compared, since no two of the packages use the same defaults. This is not surprising since defaults can be idiosyncratic. What is surprising is that four of the seven packages could not estimate Model I, because they do not provide the necessary options. This is particularly noteworthy in the case of Package X1, the documentation for which makes the claim that the user can supply all the necessary starting values. We subsequently telephoned the vendor and received confirmation that the documentation is incorrect: the user can supply starting values for the coefficients, but cannot supply the initialization. As has been discussed, all four of these packages are of little use for serious research.

The remaining three packages (X4, X5 and X7) can be evaluated using the FCP Benchmark. The points of evaluation are the accuracy of the coefficients and the accuracy of their standard errors. We could simply present the estimates for each package and let the reader compare them to Table 3. Ambitious readers would then count the number of accurate digits, perhaps noting them in the margin for later comparison. There is an easier way to present the same information.

A useful way to measure the number of digits of accuracy is to use the

pack-	par-	coef-			standard error		
age	ameter	ficient	H	OP	QMLE	IM	BW
X4	μ	3.6		2.8			
	α_0	2.3		2.5			
	α_1	2.4		2.3			
	β_1	3.6		3.2			
X5	μ	2.6		5.1		5.1	4.7
	α_0	4.8		4.6		4.6	4.5
	α_1	5.0		4.8		4.9	6
	β_1	5.5		4.6		4.7	5.0
X7	μ	6	6	6	6		
	α_0	6	6	6	6		
	α_1	6	6	6	6		
	β_1	6	6	6	6		
X4a	μ	2.6	3.1	4.9	2.8		
	α_0	4.2	4.2	4.3	4.2		
	α_1	4.4	4.4	4.3	4.6		
	β_1	5.0	4.4	4.5	4.5		

Table 4: accuracy of estimates

log relative error

$$\text{LRE} = -\log_{10}(|x - c|/|c|) \tag{6}$$

where x is the estimated value and c is the benchmark value. This measure is accurate only if the benchmark value and the estimated value differ by a factor of less than two, so that at least the first digits agree. In two circumstances the LRE is reported as zero: if the first digits do not agree, or if the computed LRE is less than unity. Each of the three packages is used to estimate Model I, and the LREs for the coefficients and standard errors are computed. Results are presented in Table 4.

It is clear that Package X7, which uses Newton-Raphson with analytic gradient and analytic Hessian, hits the benchmark precisely. Package X5, which uses a quasi-Newton method with an analytic gradient, performs reasonably well, though the coefficient on the intercept is bit less accurate than the other coefficients. Package X4, which uses Newton-Raphson with an analytic gradient with a numerical Hessian, achives less success than the other packages. However, this developer has since upgraded its GARCH procedure, and now reports the results given under X4a – a significant improvement.

6 Conclusions

During the past twenty-five to thirty years that forms the period during which econometric software has become widely available and commonly used by economists, there has been considerable innovation in both econometric techniques and the implementation of those techniques in software. In general, the focus of software evaluation, mainly in the form of reviews, has been upon this demonstration of new ideas. Much less attention has been paid to issues such as the numerical accuracy of programs or normative considerations, such as what features an econometric software package should offer its users. However, one of the implications of the results that we have presented is that even in the case of a relatively standard specification, a large variation in numerical results can be observed. Although it is not possible to draw general inferences from this case, two points should be made. First, that the results we have presented were obtained from an entirely arbitrary choice: we did not perform a series of tests of a variety of econometric procedures; we have made only one set of tests and this paper describes our findings. Second, as elsewhere noted by McCullough (1997) and Renfro (1997), this paper represents an unusual contribution: during the whole of the past 30 years, there has been surprisingly little testing done by economists of the numeric accuracy of econometric packages or of the suitability of those package for the application to which they have been put. Thus we cannot rule out the possibility that a series of benchmark tests, considering a variety of procedures, could expose a wide range of problems. For this reason we encourage others to join in the numerical and more general evaluation of econometric software procedures.

In presenting our results, we have been concerned not simply to offer a benchmark, but to try to demonstrate how the design of a software package can make a difference in terms of the usability of the software. As discussed, particularly in the case of nonlinear techniques, it is not sufficient to simply present a single set of results as definitive. Software users need to understand that nonlinear techniques raise the possibility of local, rather than global convergence, as well as a series of potential problems relating to the specifics of the algorithmic implementation. We have discussed the distinction between analytic derivatives and finite approximations, and have noted that this raises numerous complexities, one of the reasons being that the structure of the problem in a particular case cannot necessarily be determined in advance. It is possible to assert that, in general, analytic derivatives are to be preferred on the basic of numeric/analytic considerations; however, someone–the user in many cases–must take the responsibility for specifying these. Nonlinear estimation packages cannot always be presented to users as "canned" procedures; this fact makes it important to begin to consider the design and development of packages as a broadly defined set of characteristics.

In the introduction, we established two questions as critical to the development of useful benchmarks: What specifications can a tested package estimate? and What ones should a package be able to estimate? The issue of which specifications can be estimated is central to the establishment of any benchmark: obviously, benchmarks are most useful to the degree that they establish a standard for existing packages. However, the issue of which specifications a package should be able to estimate is much more centrally normative and transcends the design of software packages. It is a general rule that users of software packages are dependent upon those packages to determine the range of estimatable models. As a consequence there is a higher degree of symbiosis between the development of econometric theory and the development of software than has heretofore generally been recognized. This paper represents an inital attempt to begin to better define this interaction between benchmarks, software standards, and econometric theory using the GARCH model as a case study.

7 Computational Details

All packages were run on a 166 Mhz Pentium under Windows 95. The FCP code was compiled using Lahey Fortran 90, with two switches invoked: -dbl, to extend all real and complex numbers to double precision; and -o0, to suppress all optimization. The FCP code was altered in the following way. In both VSGARCMX.FOR and GSGARCOV.FOR there is a function VALUNC which is amended twice. First, the function is declared as REAL*8 so "FUCNTION VALUNC(IDYNAM,C,..." becomes "REAL*8 FUNCTION VALUNC(IDYNAM,C,..." Second, the value of LOG(SQRT(2*PI)) is given only to six digits, which makes LogL accurate to only that many digits. Therefore, about four lines before the end of VALUNC the constant 0.9189385 is changed to 0.918938533204672 5D0

8 Appendix: Some Numerical Details of Nonlinear Estimation

Many econometrics texts provide good theoretical discussions of the various nonlinear estimation methods, *e.g.*, Davidon-Fletcher-Powell (DFP) or Broyden-Fletcher-Goldfarb-Shanno (BFGS). Invariably, these texts mention that two different algorithms might not be able to solve the same problem. Just as invariably, they do not mention that when two algorithms can solve the same problem, one solution might be decidedly better than the other. The purpose of this appendix is to give the reader some idea of why this can occur. In particular, we discuss some numerical sources of inaccuracy in nonlinear estimation, and various ways a nonlinear procedure can fail. We draw a distinction between between an algorithm and its implementation. The algorithm is a set of rules, and this is what usually is presented in econometrics texts. How these rules are coded in a programming language is the implementation, and it is this with which we are concerned. It shall be apparent that two different implementations of the same algorithm do not necessarily yield the same answer.

Good general introductions to the subject of nonlinear maximization include the contributions by Bard (1974), Goldfeld and Quandt (1972) and Quandt (1983). See, for example, Bard (pp. 83-88) for a good presentation of the basic concepts of unconstrained optimization that underlie the discussion here. Considering GARCH estimation, maximization of the appropriate (conditional) likelihood function, $L(\theta)$ can be algorithmically implemented in many ways, but in the end usually involves iteration, particularly in the context of nonlinear maximization. Let $\theta = [\theta_1\theta_2 \ldots \theta_p]'$ be the argument of the likelihood function and let θ^0 be a vector of starting values. The iterative process

$$\hat{\theta}^{k+1} = \hat{\theta}^k - \lambda_k d_k \qquad k = 0, 1, \ldots \tag{7}$$

can be shown to be a member of a very broad class of algorithms, where d_k is the direction moved and λ_k is the distance moved on the k-th iteration (Quandt, 1983, p. 717).

As a general rule, until terminated, the iteration process involves successively changing $\hat{\theta}$ by an amount and a direction. The amount of change, λ_k, is usually termed the "step size". In a multivariate context, the direction of change, d_k, is represented by a vector, possibly representing the gradient and/or Hessian of the objective function, which vector is normally chosen so as to conform with a particular negative definite matrix. Various methods differ is the ways which λ_k and d_k are computed. The change from the k-th to the $(k+1)$-th iteration is evaluated using a "stopping rule." If this test is successful and iteration is halted, then under ideal circumstances the program should report that it has found a stationary point and upon further (successful) testing in terms of second and any higher order conditions ("convergence criteria"), that it has found a maximum. In Section Four the FCP program was described as possessing both a stopping rule and a test for successful convergence. Frequently, discussions of nonlinear estimation conflate the stopping rule and convergence criterion, treating them as one and the same.

Note that if this secondary testing is unsuccessful, the program may report that it has "stalled" – stopped at a point that is not a maximum. As discussed futher below, some packages do a poor job of conducting this examination, and consequently have a marked tendency to report having found a maximum when, in fact, they have not. Stalling can also occur when the program reaches a point from which either an acceptable step or direction cannot be determined. In this case, the proram will report, "unable to improve current iterate – iterations terminated."

8.1 Step Length

With respect to determining a good step-length, the requirements are simple: it must not be too large, it must not be too small, and it must be in a good direction. If too large, the procedure can easily miss a stationary point by stepping over it. If too small, the procedure might never get to a stationary point: imagine trying to walk up a hill by taking successively smaller steps; your path might well converge to a point which is only part-way up the hill. Now, assume a good direction, $\tilde{d}$, has been found. Then one common method of determining the step-length is to find λ^* which minimizes

$$L(\theta_k + \lambda^* \tilde{d}) \tag{8}$$

and then set $\lambda_k \equiv \lambda^*$. There are many other ways to solve this problem, many of them based on the concept of a "line search" – admissible values for λ are restricted to some interval, and this interval is searched for the value which minimizes (8). See Dennis (1984) for a discussion of step-length determination. We defer consideration of finding a good direction until later.

8.2 Stopping Rules and Convergence Criteria

Let $g(\theta)$ be the gradient of the likelihood function. In principle, iteration should continue until $g(\theta) = 0$ for some $\theta = \hat{\theta}$, which is, of course, the requirement for a stationary point. Then $L(\hat{\theta})$ can be investigated to determine whether $\hat{\theta}$ is a maximum. However, given the nature of finite precision computation, it is a far too stringent test to impose the operative requirement that $g(\theta) = 0$ exactly. Bear in mind that decimal numbers are represented in the computer by finite-precision binary numbers, which are essentially approximations. Consequently, it is a rare event that a computation yields the number zero exactly. Thus it is appropriate to evaluate the results of computations on the basis not that they evaluate to exactly zero, but rather approximately zero, *i.e.* $-\epsilon < g(\theta) < \epsilon$ for some small value, ϵ, called the "tolerance." Clearly this condition admits not only stationary points, but nonstationary points for which the gradient is near but not equal to zero. Thus, even locating a stationary point is fraught with the potential for numerical error.

It can be shown (Bard, p. 86-88) that, in principle, once a stationary point is found succesive values of θ^k should be approximately the same (which is the definition of "stationary"). Therefore, the change from one iteration to the next should be "essentially" zero; that is, it should be smaller than epsilon upon testing using another stopping rule $|\theta^{k+1} - \theta^k| < \epsilon$.[4] Suppose, though, that the step-length rule happened to produce an exceptionally small λ_k so that θ^{k+1} was close to θ^k Then yet another rule would be necessary to determine whether θ^{k+1} really was a stationary point, and still the convergence criterion must be applied to determine whether the stationary point is an extremum. This demonstrates why stopping rules and convergence criteria should not be conflated. In fact, there are many stopping rules. Some common ones are

1. $k > K$ where K is an upper bound on the number of iterations; "maximum iterations reached"

2. $|L(\theta^{k+1}) - L(\theta^k)| < \epsilon$; "function convergence"

[4]This is yet another reason that a step length should not be "small." If the step length could be arbitrarily small, then this stopping rule could always be satisfied far from the maximum.

3. $\max_i[|\theta_i^{k+1} - \theta_i^k|] < \epsilon \qquad i = 1, 2, \ldots, p$; "parameter convergence"

4. $\max_i[(\theta_i^{k+1} - \theta_i^k)^2] < \epsilon \qquad i = 1, 2, \ldots, p$; "squared parameter convergence"

5. $|g(\theta)| < \epsilon$; "gradient convergence"

Of course, the Hessian can be examined if it is available (not all algorithms require the calculation of second derivatives). These criteria can be used singly or in combination, frequently the latter because any one does not necessarily imply any other. When convergence is achieved, set $\hat{\theta} = \hat{\theta}^{k+1}$; the procedure has converged in $k + 1$ iterations.[5] Ideally, $\hat{\theta}$ is a stationary point. It may or may not be a local extremum, and so $L(\hat{\theta})$ must be examined carefully to determine whether or not it is a maximum. Some of these criteria may be dependent upon the scaling of the variables. Another criterion which merits mention is "relative offset convergence criterion" discussed in Bates and Watts (1988, ch. 2). See also Belsley (1980), Dennis, Gay and Welsch (1981), and Gay (1983) for further discussion.

8.3 Derivatives

To see the difference between numerical analytic derivatives, consider the forward-difference approximation to the first derivative to a univariate function, $f'(x) = (f(x+h) - f(x))/h + R(h)$ where $R(h)$ is the remainder. Note that $R(h)$ might not be negligible if the function is highly nonlinear. Exact analytic derivatives are not contaminated by this approximation error, $R(h)$. This approximation error can be decreased at the expense of computational time by using a more sophisticated form of numerical derivative,

[5]The information in d_k usually provides the standard errors for the coefficients. Note that at this point, the subscript on d is k, so if d is used to compute standard errors, one more iteration must be made to update d to $(k + 1)$.

such as the central difference: $f'(x) = (f(x-h) - f(x+h))/2h + R(h)$ [check this formula]. Of two otherwise identical programs, one which uses forward differences and another which uses central differences, the latter can be expected to be more accurate. In some cases, though, an algorithm based on the most accurate numerical first derivatives can stall in a region where $||g(\theta)||$ is small but non-zero, whereas a similar algorithm based on analytic derivatives will encounter no difficulty. Since convergence tests (as opposed to stopping rules) are based on derivatives, *cet. par.,* inaccurate computation of derivatives can lead to false convergence results. Sometimes the procedure will fail to recognize a maximum it has encountered but, more commonly, will label as a maximum a point which is not even stationary.

In the use of either forward or central differences, there is an optimal choice of h. If h is chosen too large, then truncation error dominates the result, while if h is chosen too small, then roundoff error and cancellation error dominate the result. See Press, *et al* (1992), for further elementary discussion. However, determining the optimal choice of h for a numerical first derivative usually entails knowledge of the second derivative, producing a Catch-22: the second derivative cannot be calculated before the first derivative. Thus, if two otherwise identical programs both use foward differences, but differ in how h is calculated, there might be some difference in accuracy (Dennis and Schnabel, 1983 §5.4). Gill, Murray and Wright (1981, p. 285) have noted, "[T]he user should be aware of the increased complexity and decreased reliability that result when exact first derivatives are not available to an algorithm." Thus, if a package uses numerical derivatives as default but accepts user-supplied analytic derivatives, the user is well-advised to take the time and trouble to supply the analytic gradient. Further discussion of numerical derivatives can be found in Gill, Murray, and Wright (1981) and

Dennis and Schnabel (1983).

8.4 Algorithm Choice

As Quandt (1983) has noted, algorithms have a variety of characteristics, not all of which can necessarily be embodied in a specific algorithm. Thus the choice of method of maximization necessarily involves tradeoffs, for there is no best algorithm and, in fact, the characteristics of any one algorithm may vary from problem to problem, being less amenable to one particular problem and more amenable to another. This is because different algorithms make different assumptions about the problem structure. For example, nonlinear least squares (NLS) problems have a special structure, and methods designed specifically for NLS are likely to be better at solving NLS problems than general unconstrained maximization routines. Likewise, some algorithms are more sensitive to the control parameters (starting values, step-length, etc.) than others. For example, Newton-Raphson is more dependent on the choice of starting values than quasi-Newton methods.

Two important considerations are: (1) the robustness of an algorithm, that is, its ability to find a potential "solution" and (2) the cost of execution. Today, with powerful microcomputers, cost is essentially a matter of time, but in the past it was defined in terms of computer resource use, especially on mainframe computers. The robustness of an algorithm can be a subtle issue, depending not only the problem at hand, stopping rules, and step length, as noted, but also on the amount of derivative information (gradient, or gradient and Hessian), and the types of derivatives (numerical or analytic). Fraley (1989) gives a lucid discussion of these issues in the context assessing algorithms for nonlinear least squares.

For the maximization of a function $F(\theta)$, a common conceptual starting point is the specification of a second-order Taylor approximation a point θ_k:

$$F(\theta) = F(\theta_k) + (\theta - \theta_k)' g_k + \frac{1}{2}(\theta - \theta_k)' H_k (\theta - \theta_k) + R(\theta - \theta_k) \tag{9}$$

where g_k is the vector of first derivatives (the gradient) evaluated at the point θ_k and H_k is matrix of second derivatives (the Hessian) evaluated at the point θ_k, which reflects the concavity property of the function to be maximized. In order for (9) to be valid, F must be analytic in a region about θ_k. Then from (7) choosing $d_k = H_k^{-1} g(\theta_k)$ and setting $\lambda^k \equiv 1$ defines the Newton method[6]

$$\theta^{k+1} = \theta^k - H_k^{-1} g_k \tag{10}$$

The Newton method critically depends for its properties on the starting value θ_0, being "close" to the value which maximizes the likelihood. The reason is that in order for H_k^{-1} to be a part of a "good" descent direction, H_k^{-1} must be negative definite. Though H_k is negative definite when θ_k is close to θ, when θ_k is not close to θ, H_k might not be negative definite and the procedure breaks down.

A variety of schemes for solving this problem of negative definiteness center around replacing H_k in (10) with some Q_k, where Q_k is constructed so as to be negative definite whatever the value of θ_k. A decided advantage of this approach is that the convergence properties no longer depend so critically on the starting values, as compared to Newton-Raphson. One particular class of such schemes, the quasi-Newton (variable metric) method, constructs Q_k using F and g. In quasi-Newton routines, an approximation to the Hessian is built over several iterations with the hope that, by the time iterations converge, $Q_k \approx H_k$. On each iteration, Q_k is updated by $Q_{k+1} = Q_k + U_k$ where U_k is the updating matrix. One rule for constructing U_k leads to the

[6]If $\lambda_k \neq 1$ in (10), then the algorithm often is referred to as a "modified Newton method."

DFP algorithm, while another leads to the BFGS algorithm. Since each iteration yields information only about one direction, Q_{k+1} can be expected to differ from Q_k by a low-rank matrix. When U_k has rank one, this is called a "rank one update." Naturally, there exist methods based on rank two updates, such as BFGS and DFP. Because these methods builds up curvature information slowly, if the procedure terminates in only a few iterations, Q_k is likely to be a poor approximation to H_k; the coefficient estimates may be good, but the standard error estimates will not. Even if several iterations occur, standard errors based on Q_k will be inaccurate to the extent that $Q_k \neq H_k$. An additional difficulty with the quasi-Newton approach and other methods which use only gradient information is that they cannot detect and move away from a saddle-point as well as Newton-Raphson can.

One common approach to securing the best of both worlds is to use a "hybrid algorithm." An algorithm which is not so sensitive to starting values, such as a quasi-Newton method, is used to move the iterate "close enough" to the solution, and the the problem is handed off to a method with superior convergence properties, such as Newton-Raphson. When both algorithms are of the gradient variety, this is called a "mixed gradient algorithm" (Dagenais, 1978).

Generally, choice of algorithm and its implementation, starting values, stopping rule, and the method of computing derivatives all interact to determine whether the program can compute the optimum and, if so, how accurately. The reasons a procedure can fail can be simply enumerated.

8.5 Ways a Procedure Can Fail

The preceding discussion makes clear several of the reasons that, when two packages solve the same problem, the solutions might not be equally accurate.

These same reasons can also shed light on the ways a package can fail to produce a solution (Murray, 1972).

There is a variety of ways a procedure can fail to obtain satisfactory parameter estimates, but these can be roughly categorized under four headings: (1) failure to converge (the iterations do not stop until the upper bound is hit); (2) failure to improve the current iterate (the procedure reaches a point from which it cannot find another step to take); (3) failure to recognize a maximum (the procedure stops at a maximum, but does not recognize that the point is a maximum); and (4) "false maximum" (the procedure stops at a point which is not a maximum but indicates that a solution has been found). In the first three cases the user has some idea that something is amiss, and so can take appropriate action. Diagnostic output can be examined to see whether the iterates are converging appropriately, etc. (Gill, Murray and Wright, 1981, §8.3) and the usual "fixes" can be applied: change the starting value, vary the options, even try another algorithm. The fourth case is a catastrophic failure, for the user has no idea that anything is wrong. We discuss each in turn.

In failure to converge, the procedure hits the upper bound on the number of iterations, the user resets the upper bound, and again the procedure hits the bound. There are two primary reasons that a procedure may fail to converge. First, the iterates may diverge monotonically or in an oscillatory fashion. A strong hint that this is occurring comes when the user must increase the maximum number of iterations. It can be confirmed by examining the function values and other diagnostics. Second, the iterates may cycle, with (nearly) the same sequence of iterates recurring. Again, the user has to reset the maximum number of iterations, and examination of function values and diagnostics reveals some sort of cyclical behavior. There is yet a third reason: the algorithm may be exceedingly slow for this particular problem, making very little progress on each iteration, perhaps requiring several hundred iterations.

That a step length and a step direction have been found does not mean that a step is taken. This is the problem underlying "failure to improve current iteration". The step must be "acceptable" before it is taken. Generally, this means that for a candidate θ^{k+1}, it must be true that $L(\theta^{k+1}) > L(\theta^k)$. If such a step cannot be found, then the program will issue a "failure to improve the current iterate" message and terminate. One common reason for such a message is that the convergence tolerance has been set too small for the accuracy to which the function and gradient are computed. This can be especially true if the likelihood is flat in the region of the maximum. Alternatively, the function or one of its derivatives might be discontinuous near the point at which the algorithm stalls. This discontinuity might be real, or it might be a numerical artifact of finite precision (see Murray, Gill and Wright, 1981, p. 327). Yet another possibility is that the derivatives might be inaccurate.

Formally, failure to recognize a maximum is a special case of failure to improve. If the program stops at a maximum but fails to recognize it as such, it is likely to finally exhibit the error message "failure to improve," indicating the inability to determine a next step that increases the value of the given objective function, yet also the inability to determine the satisfaction of the second order conditions. A simple test which often works in practice is to increase the convergence tolerance. If the procedure converges to the same point but this time labels it a maximum, it probably is a maximum.

"False maximum" occurs when the procedure stops at a point which is not stationary, and this point somehow satisfies the program requirements for the point to be labelled a maximum. This problem often arises due to poor implementation of the termination criteria (Gill, Murray, and Wright, 1981, p. 306). Careful examination of the diagnostics might indicate that the point is not a maximum, but even then there is no "fix" which the user can apply using that same software package. For this reason a package should err on the side of caution: better to label a maximum as "no solution found" than to label a non-solution as a solution. Some packages have a marked tendency to report false maxima, and users contemplating nonlinear estimation should avoid the use of such packages.

References

Bard, Y. (1974), *Nonlinear Parameter Estimation*, New York: Acadmeic Press

Bates, D. M. and D. G. Watts (1988), *Nonlinear Regression and Its Applications*, New York: J. Wiley and Sons

Belseley, D. (1980), *On the Efficient Computation of Full-Information Maximum-Likelihood Estimation*, *Journal of Econometrics* **14**, 203-224

Bollerslev, T. (1986), "Generalized Autoregressive Conditional Heteroscedasticity," *Journal of Econometrics*, **31**, 307-327

Bollerslev, T. and E. Ghysels (1996), "Periodic Autoregressive Conditional Heteroscedasticity," *Journal of Business and Economic Statistics*, **14**, 139-151

Bollserlev, T., R. F. Engle and D. B. Nelson (1994), "ARCH Models," in R. Engle and D. McFadden, eds., *Handbook of Econometrics, volume 4*, Amsterdam: North-Holland, 2959-3038

Calzolari, G. and L. Panattoni (1988), "Alternative Estimators of FIML Covariance Matrix: A Monte Carlo Study," *Econometrica* **56**, 701-714

Dagenais, M. G. (1978), "The Computation of FIML Estimates as Iterative

Generalized Least Squares Estimates in Linear and Nonlinear Simultaneous Equations," *Econometrica*, **46**, 1351-1362

Dennis, J. E. and R. B. Schnabel (1996), *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Philadelphia: SIAM

Dennis, Gay and Welsch (1981) "An Adaptive Nonlinear Least-squares Algorithm," TOMS 7, 348-368

Dewald, William G., Jerry G. Thursby and Richard G. Anderson. 1986. "Replication in Empirical Economics: The *Journal of Money, Credit and Banking* Project," *American Economic Review*, 76:4, pp. 587-603.

Diebold, F. X. and T. Schuermann (1998), "Exact Maximum Likelihood Estimation of Observation-Driven Econometric Models," in R. S. Mariano, M. Weeks and T. Schuermann, eds., *Simulation-Based Inference in Econometrics: Methods and Applications*, Cambridge: Cambridge University Press, ????-????

Donaldson, J. R. and R. B. Schnabel (1987), "Computational Experience with Confidence Regions and Confidence Intervals for Nonlinear Least Squares," *Technometrics* **29**, 67-82

Engle, R. F. and T. Bollerslev (1986), "Modelling the Persistence of Conditional Variances," *Econometric Reviews* **5**, 1-50

Feigenbaum, S. and D. M. Levy (1993), "The Market for (Ir)Reproducible Econometrics," *Social Epistemology*, **7**, 243-244

Fiorentini, G., G. Calzolari and L. Panattoni (1996), "Analytic Derivatives and the Computation of GARCH Estimates," *Journal of Applied Econometrics* **11**, 399-417

Fraley, C. (1989), "Software Performance on Nonlinear Least- Squares Problems," Report No. 89-1244, Stanford University Department of Computer Science

Gay, D. M. (1983), "Algorithm 611: Subroutines for Unconstrained Minimization Using a Model/Trust-Region Approach," *ACM TOMS*, **9**, 503-524

Gill, P. E., W. Murray, and M. H. Wright (1981), *Practical Optimization*, New York: Academic Press

Greene, W. (1993) *Econometric Analysis, 2e.* New York: MacMillan.

Longley, J. W. (1967), "An Appraisal of Computer Programs for the Electronic Computer from the Point of View of the User," *Journal of the American Statistical Association*, **62**, 819-841

Lovell, M. C. and D. D. Selover (1994), "Econometric Software Accidents," *The Economic Journal*, **104**, 713-726

Lumsdaine, R. (1996), "Consistency and Asymptotic Normality of the Quasi-Maximum Likelihood Estimator in IGARCH(1,1) and Covariance Stationary GARCH(1,1) Models," *Econometrica* **64**, 575-596

McCullough, B. D. (1997), "Benchmarking Numerical Accuracy: A Review of RATS v4.2," *Journal of Applied Econometrics*, **12**, 181-190

Murray, W. (1972), "Failure, the Causes and Cures," in W. Murray, ed., *Numerical Methods for Unconstrained Optimization*, New York: Academic Press, 107-122

Press, W. H., S. A. Teukolsky, W. T. Vetterling, and B. R. Flannery (1994) *Numerical Recipes in Fortran, 2e* New York: Cambridge University Press

Quandt (1983)

Renfro, C. (1997) "Normative Consideration in the Development of a Software Package for Econometric Estimation," *Journal of Economic and Social Measurement*, **23**, 277-330

Silk, J. (1996) "System Estimation: A Comparison of SAS, SHAZAM, and TSP," *Journal of Applied Econometrics*, **11**, 437-450

Ueberhuber, C. W. (1997), *Numerical Computation, vol. 2*, Berlin: Springer